Nicholas Chan
nipchan@ucsc.edu
4/18/2021
CSE13s Spring '21

Design Document:
Assignment 2: Math Library

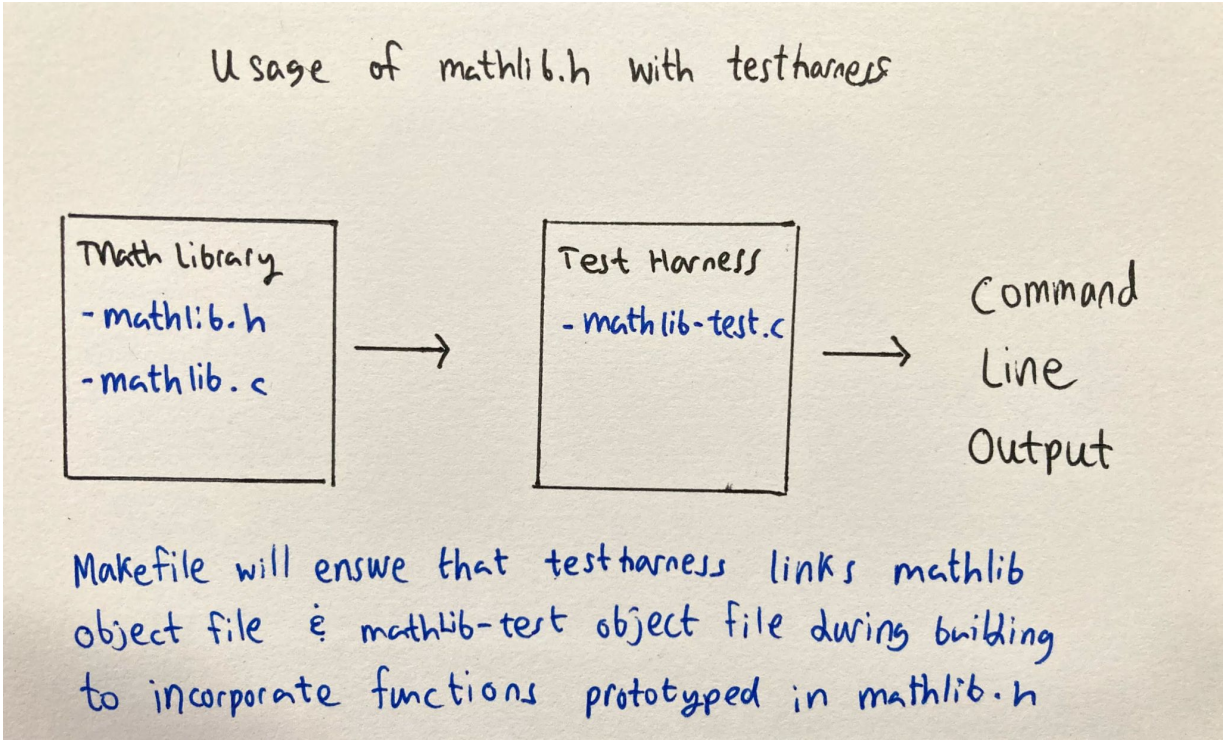
Trigonometric and logarithmic functions are a group of functions that are difficult to graph and calculate by hand. However, through methods of approximation and computer computation, we can obtain results very close to the true results of such functions.

In assignment 2, I have been given the task of creating a math library containing functions for computing arcsin, arccos, arctan, and log. As these mathematical operations have been well studied, I have learned that they can all be approximated using Newton's Method. My implementation of the functions for assignment 2 will rely heavily on Newton's Method and loop control. The main moving parts of this program will be:

- The main functions for arcsin, arccos, arctan and log
    - Will be found in my math library file
- My testing harness to compare the results of my functions with the results of the functions provided by the math.h library
    - Will be a file separate from my math library file
- The ability for my test harness to take in command line arguments
    - This feature will be used to provide a comparison between my implementation and math.h's implementation of the main functions of this assignment

Top Level

How my Files will Interact



1. Functions for arcsin, arccos, arctan and log will be stored in mathlib.c
2. mathlib.h contains the prototypes of the functions found in mathlib.c
3. mathlib-test.c will include the mathlib.h library and use the functions prototyped in it to run tests to compare my implementation of the functions with the implementations of math.h
4. Output will consist of a table printed out in the command line

Pseudocode

Note:
With the Newton method, the method of approximation for all these functions is practically the same)

mathlib.c file:

Preprocessor:
Define EPSILON as 1E-10

Static Functions:
Static function for Exp
Static Function for Sqrt
Static Function for absolute value

Functions:
Functions for arcsin, arccos, arctan, log base e

General form that my functions implementations will take using Newton's Method:
Function (Takes in double, call it "x") {
[Old = a number inside the range of the function we are approximating]
[New = Old]
[Diff = Old]
(while absolute value of Diff > EPSILON)
F(x) = Inverse of function
F'(x) = Derivative of inverse of function
New = Old - F(x)/F'(x)
Diff = New - Old
Old = New
Return value of new once Diff =<EPSILON}

Testing Harness:

Preprocessor:
Options for command line arguments: -a, -s, -c, -t, -t

Static Functions:
Static function for printing headers (string of my function name)

Main Function:
Bools associated with each valid command line argument.

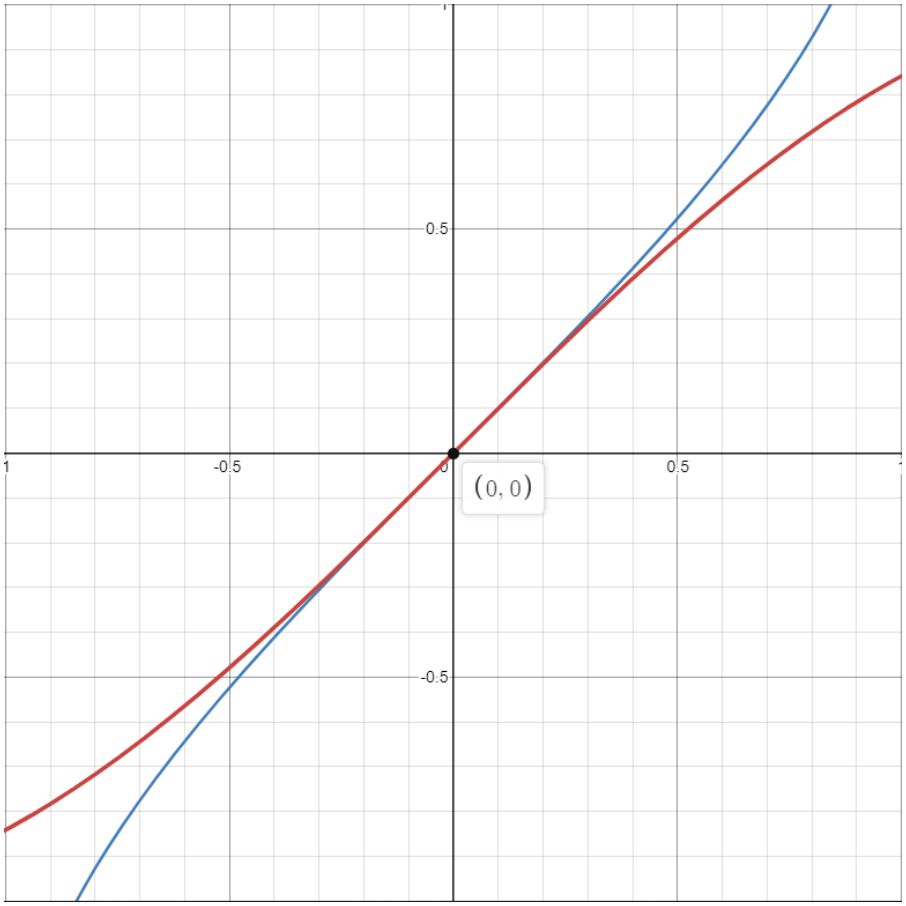
Switch statements for assigning bools to associated arguments given each argument typed on the command line when the test harness is run.

Conditional statements for printing tables after parsing of command line arguments from switch statement.

Usage of Newton's Method

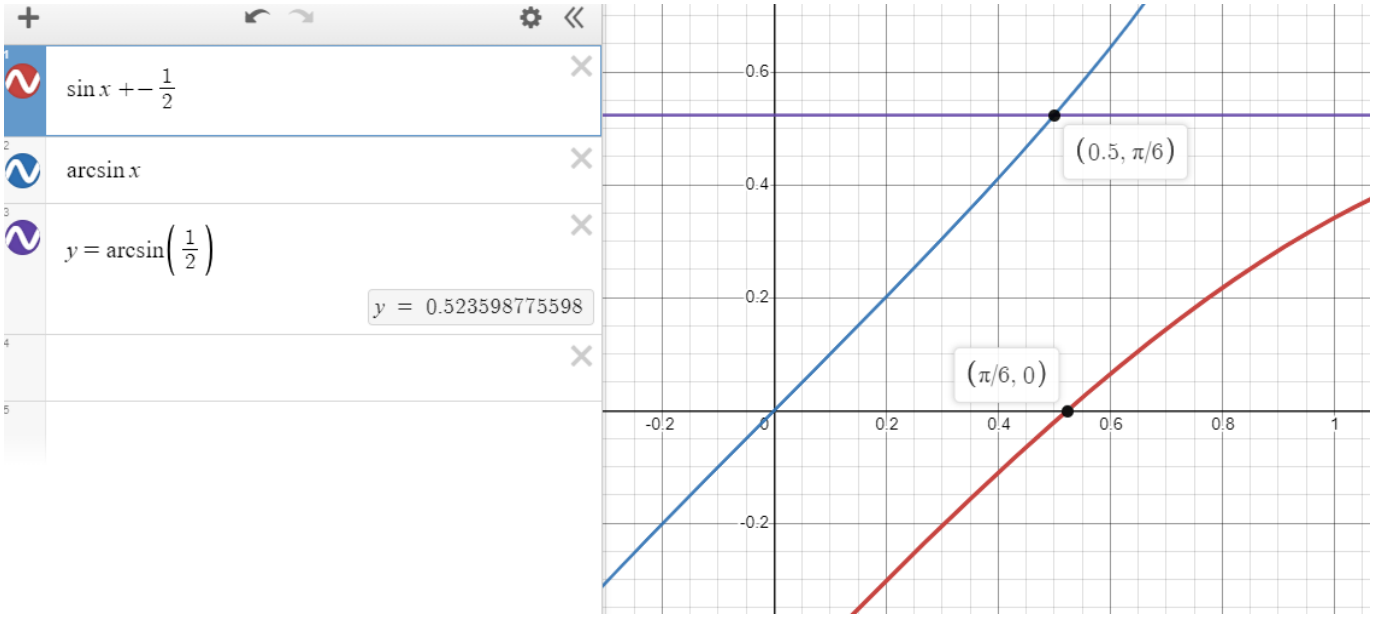
In assignment 2, I have used Newton's method for finding the roots of functions. Since roots are just the value of a function's independent variable that makes the function zero, Newton's method won't have great use immediately. To utilize the power behind Newton's method, I must transform the function that will be acted upon with Newton's method by the number entered as input.

I will be using the approximation of **arcsin(x)** with the iterate of **new = old - function+input/1st derivative of function** as an example.



arcsin(x) shown in blue, **sin(x)** in red

The root of **sin(x)** is zero. Coincidentally, with **sin(x)**'s root of **x = 0**, the value **arcsin(x)** when **x = 0** is also **0**. We find an even more interesting result when we translate **sin(x)** down by **½** units .



This shift of **sin(x) - ½** yields a root of **x = pi/6** which we can find using Newton's method. This same root of **x = pi/6** when plugged into **arcsin(x)** yields **½** which we know to be our **original input**. This relationship between shifting a general function **F(x)** by some **input** and the resulting value of its inverse is a characterization of the inverse method of approximation. This is the main idea underlying my implementation of assignment 2's functions with Newton's Method.

Design Process

Prior to starting the programming process of this lab, I had to review the concept and use of Newton's Method from the clues provided by the assignment's description. I also watched a few videos which I reference at the end of this doc to help me visualize how I could translate graphs to make use of Newton's Method.

1. To begin the programming process, I first made a small test file where I could implement Newton's Method for first the arcsin approximation. I wrote out the general formula I used which used a Newton's iterate of the form:
    a. (old) - (the inverse of the function I am approximating minus the vertical shift *note: vertical shift = input of function*) / (the first derivative of the inverse of the function I am approximating.
2. After completing my first function with Newton's Method (arcsin), the rest followed a similar process.
3. Next I developed the mathlib-test harness. The test harness heavily relied on the getopt() function for parsing command line options
    a. The use of getopt() was tied to a switch statement that would set bools corresponding to the functions in my mathlib library.

    b. These bools would then indicate to a block of conditionals which functions to print tests out for
4. Finally I created a Makefile to properly compile the source code of mytest harness mathlib-test.c and library mathlib.c
    a. This called for -lm to link the object files of these two files to successfully use the functions prototyped in the mathlib.h header file, which was included in mathlib-test

The design of the individual math functions was fairly straightforward with a reasonable amount of math knowledge required to make each of them work. I felt like I could have done more practice with parsing command line options to make the prints for error catching in mathlib-test less wasteful. I also feel that I should have practiced using the -lm argument for compiling programs with libraries a little more to make the process of implementing it in this assignment smoother. If I could change something in this lab, I would have made an option in mathlib-test to write out easily parsed text files containing x values and the results of comparing my implementation of math functions with the math.h library's implementation of math functions. This would make graphing the data from these tests easier.

Significant References:

- https://www.youtube.com/watch?v=fIhggF0blKY
- https://www.youtube.com/watch?v=1uN8cBGVpfs
- https://www.youtube.com/watch?v=3d6DsjIBzJ4